Operating Systems Project 3

Alejandro Lopez

Nova Southeastern University

**Table of Contents:**

[Description](#_Description)

[Design](#_Design)

[Client-Server System: Presentation Layer (Client)](#_Client-Server_System:_Presentation)

[Client-Server System: Processing/Application Layer:](#_Client-Server_System:_Processing/Ap)

[Client-Server System: Data Layer](#_Client-Server_System:_Data)

[Implementation](#_Implementation)

[Communication System: Client-Server](#_Communication_System:_Client-Server)

[The Client: Presentation Layer](#_The_Client:_Presentation)

[Cache Service: Holding the data elements](#_Cache_Service:_Holding)

[The Server: Receiving requests from the client](#_The_Server:_Receiving)

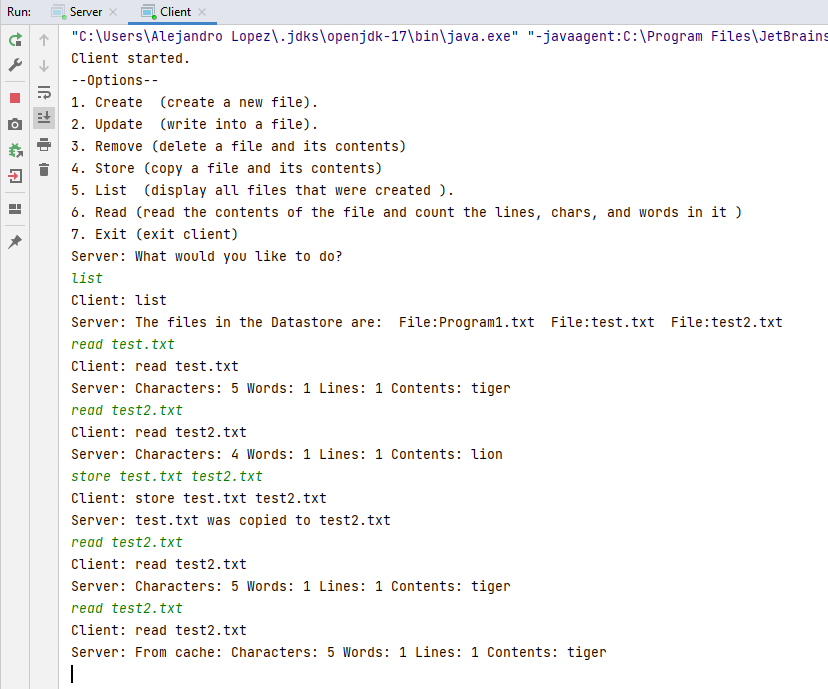
[How to install and use my system](#_How_to_install)

# **Description**

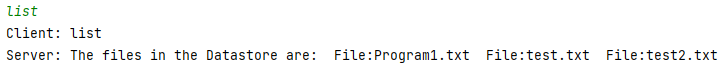
This is a client server program which provides word count support and offers a multi-tiered application. This program supports multiple client connections through a multi-threaded design and includes a separate tier where all the persistent data is stored. The word count system has fixed-size cache implemented in it meaning that the file for which the contents were counted frequently will be stored in cache while those files whose words were least frequently counted are removed from the cache.

# **Design**

## **Client-Server System:** Presentation layer (Client)



* *Overview*:
  + Only if the server is running, multiple client instances can be initiated. The client instances can be controlled through commands which are displayed to the user at the beginning of a client instance. The client displays the what the user requested, the client receiving the user input, and the server returning the requested job completed.
* *Available commands (As shown on the image):*
  + Create, Update, Remove, Store, List, Read, Exit
* *User input:*
  + There are 3 types of user input on a client instance:
    - Single input:
      * Example:



* + - * These commands include list and exit because they do not require additional input to function
    - 1 division (two inputs):
      * Example:

Text

Description automatically generated

* + - * These commands include create, remove, and read. These types of commands require the initial function followed by a space and then the file name which will be worked on. For example: read test.txt. The file name cannot be inputted in a new line and after typing read for example, the file name must follow after a space.
    - 2 divisions (three) inputs:
      * These commands include store and update.
        + Example:

Text, letter

Description automatically generated

* + - * + Update:

An example: update test.txt hello world. The type of input which contains ‘update’ writes new contents inside a file. First the command, then the filename, finally the new contents for the file. However, while the program can indeed read a file with multiple lines and count the number of lines, it cannot edit a file and add contents that have multiple lines in them. The program can only modify a file to have contents of one line.

* + - * + Store:

An example: store test.txt test2.txt. This type of input copies the contents of a file into another file. The requirements for input are store, the source file name, finally the destination file name. Again, each input will be inputted in the same line separated by a space.

* + *Incorrect input (Exception Handling) Scenarios:*
    - First input file does not contain the format .txt
    - Second file does not contain the format .txt
    - The user input does not match any of the commands
    - The file already exists (creating a file)
    - The file does not exist (any command related to editing a file)
    - When copying a file, the user cannot copy a file into itself
    - Spaces are missing between commands
    - There is a command but no filename
    - There is a command and a filename, but the third input is not there (update and store)

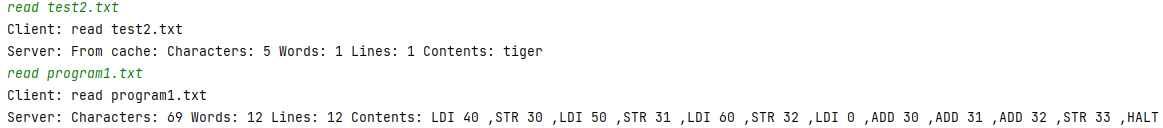
## **Client-Server System:** Processing/Application Layer:

Text

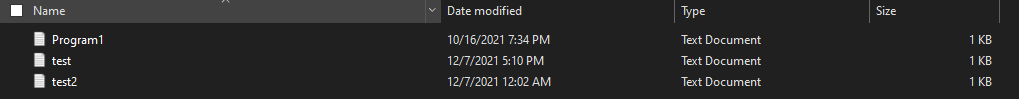
Description automatically generated

Text

Description automatically generated

* *Cache Service:*
  + How it works:
    - When a client instance reads a file more than once then the contents of that file will be stored inside the cache. Now instead of showing the contents directly from the file, the information gets displayed from the cache. However, the cache is limited to a set size of 5. If the number of file contents stored inside the cache exceeds 5 then the least used item will be removed from the cache.
  + Display(Client side):
    - As stated above, once the file gets read more than once by the user’s commands then the client will start displaying the file’s content from the cache instead of from the file iteself.
  + Display(Server side):
    - The server will display the cache’s current status meaning how many files are sinde the cache, the least used item and when it was last used, and checking if a file is in cache and if it is not then add it to the cache. If the cache capacity does reach a number more than 5 then the server will display which file got removed.
* *Word Count Server:*
  + How it works:
    - When the user inputs their command then the client sends that command to the server to complete the job and once the job is completed, the server displays the finished product. For example if the user wants to read test.txt then the client will send that request to the server which will use the logic inside the protocol to complete the job and then display the contents, the word count, the character count, and the line count of test.txt.
  + Line count:
    - The server is able to count all the lines inside a file and display them, however, the lines will not be displayed by the client. Instead, lines will be replaced by commas. An example of this is when program1.txt which is a file with 12 lines is read. As showed in the image, each new line is replaced by a coma.
  + Cache:
    - As mentioned in the section dedicated to cache, when a file’s contents are read more than once then they will be stored on the cache. If a file is stored in cache then the contents will not be displayed and couinted from the file, instead, they will be showed from the cache.

## **Client-Server System:** Data Layer





* *File and Word Count Data Persistent Store:*
  + Persistent Store:
    - The data layer consists of the files that are being used and the cache which is storing them. Various commands manipulate a directory in the user’s local machine where the user can create, edit, delete, copy, and read those files. Each time a new file is created it goes to the hard coded path, therefore, it can be said that the data store is the folder where the files are held.
  + Word Count:
    - In addition to reading the files content’s, the user can count the number of lines, characters, and words inside a particular text file if it exists in the data store.
  + Cache:
    - The only scenario where they system would benefit from using cache would be when reading a file because create, update, and remove all either remove the contents of the file, modify it, or simply creates a new file. To store something in the cache means that it is something that will be done often with the same file, therefore, if a file is read multiple times with the same content, then it should be saved in the cache to save the system from having to open the file and read its contents as well as count them.

# **Implementation**

**Diagram

Description automatically generated**

## **Communication System:** Client-Server

* *Overview:*
  + The Socket class is in the java.net package, therefore, java.net.\* is being used within this project. This project works using localhost, which uses the default local computer IP address of 127.0.0.1. This way, both the server and the client will be running on the same computer.

Text

Description automatically generated with medium confidence

## *Server:*

* + The key of this program is inside the try catch section of the code. The clientSocket instantiation of a socket is what lets the server listen to the given port which in this case is 3307. The server is automatically set up at the computer on which it is running and the clientSocket instantiation on the next line uses the accept() method of Server Socket. This method waits until a client attempts to connect to the server, and it returns an instance of the socket class. The socket instance is how the server communicates with the client. clientSocket.getOutputStream() returns the output stream through which the server can talk to the client, and clientSocket.getInputStream() returns the input stream through with the server can hear the client. A print writer is then created to using the output stream from clientSocket to output and it sends the data to the client in a print out.

Text

Description automatically generated

* *Client:*
  + Just like the server, the most important part of the client is located inside the try catch block of code. A server connection is attempted by creating an instantiation of the socket class. This instantiation tries to contact the server at local host through the port 3307, which is the same port where the server is listening. Once the socket is set up, it works just like the server instantiation socket. Then, the input stream is obtained and the data is obtained through the getOutputStream() to then display the data.

## **The Client:** Presentation Layer

Graphical user interface, text, application, email

Description automatically generated

* *Application/Presentation Layer:*
  + Overview:
    - This area of the program contains the logic for storing, updating, removing, counting, and reading files. This class is made up of classes that control each command which would manipulate a file and in all of them the Java libraries File and File reader are required. A new instantiation of a file from the File library is created on each method. The next step is checking if the file already exists which is done through the exists method which uses the exists() method from the class File. The method in list functions similarly to exists with the difference that in list returns true or false.
  + Create:
    - If the file does not exist then the new instance of the file java library is used to create a new file using createNewFile().
  + Update (Write):
    - This method uses the create method to create a new file with the same name but with the desired contents of the user. This is done through a new instance of a Print writer using its function called print().
  + Read:
    - A new instance of a string builder and a file is created. Then if the file exists then a new instance of a buffered reader is made to read the contents of the specified. Using a for loop to check if there are still more lines to be read with the buffered reader instance, all the file’s contents get stored on a separate string to return that string and show it on the client.
  + Remove (Delete):
    - If the file exists use the method from the java file library called delete() to remove the file from the data store.
  + Store (Copy Content):
    - This method checks if the file exists and gives that value to a Boolean. If the Boolean returns true and the specified file name does indeed match a file then return true, else if it is in the directory return true, else return false because the file does not exist.
* *Protocol:*

Text

Description automatically generated

* + Overview:
    - The protocol is also connected to the server but through the client and the protocol’s functionalities all get displayed on the client, therefore, grouping the protocol with the client is reasonable. The protocol takes care of all the logic for the user’s input to determine which action should be taken in the client, the server, and the cache.
  + States:
    - The protocol has two states which are WAIT and FILES. These two states have the values of 0 and 1 respectively. If the state is WAIT then both the client and the server have to wait for the user to input a new command. Now, if the state is FILES then that means that the user already submitted a command and the program can now perform an action with the files.
  + User input
    - How it works:
      * The user input is processed in the protocol, but it is taken in by a protocol instance inside the server.
    - Processing the input:
      * This is the protocol’s main job together with entering file contents into the cache. The input which comes from the server’s protocol instance gets broken up either into a string array of 2 or a string array of 3. This depends on if the command specified by the user is either update or store. However, independent of how many times the string gets divided the string will always need to have a command. This command gets stored inside a new variable called choice which will be used to perform each task.

Graphical user interface, text, application

Description automatically generated

* + - * Update and Store:
        + these two commands are managed through an if statement to check if the string contains either of the two and if it does then the string gets split into 3 whenever there is a space in the input. The first index of the string array contains the command, the second contains the file’s name and the third includes either the second file’s name (store) or the new contents of the file(update).

Text

Description automatically generated

* + - * List and exit:
        + These two require no string division and therefore the choice variable is the only one that needs to be checked for to perform those tasks.

Text

Description automatically generated

* + - * The rest of the commands:
        + The input string gets divided into two parts which are the choice and the filename to be used for the task.

Text

Description automatically generated

* + - Deciding which task to complete:
      * The class protocol is comprised of many else if cases to decide which task will be completed once the user input was processed. After the if statements identify which choice the user selected and after the input gets checked for the wrong input then the Application class is called which includes all the methods to modify a file.
* *Word Count System:*

Text

Description automatically generated

* + Overview:
    - The logic for this system is located inside the protocol class in the section of the code where the input was already processed by the if statements to determine that the user wanted to read the file. Then the program is able to call Application.read() which reads the contents of the file and we can start counting its contents. This part of the code is related to the cache but that will be gone over in the next section. It is related because this is where it is decided whether the file’s contents will be stored in the cache.

## **Cache Service:** Holding the data elements

Text

Description automatically generated

* *Data Structures used:*
  + Hash map: A data structure that stores items in pairs through the “key/value” architecture, and one can access them by an index of another type.
    - Why it is being used:
      * This is helpful because in this case the key can be the file’s name and the value can be the file’s contents as well as its word count.
  + Priority queue: In priority queue the items are ordered by a key value so the item with the lowest key is at the front and the item with the highest key is at the rear. The lower the value, the higher the priority.
    - Why it is being used:
      * We can use this data structure because the cache’s size is supposed to be limited and the cache is a system where the least used value gets removed.
* *How I implemented the Cache:*
  + This implementation has two classes:
    - Element.java:
      * This is the place holder for values in the priority queue (PQ) which has a time stamp and the value attributes. The time stamp is updated whenever a new element is placed inside the Cache.
    - Cache.java:
      * This class uses the priority queue and the elements are ordered inside this data structure based on the time stamp. The time stamp determines which is the least used item since it records when was the last that a certain element was accessed. Whenever a least used element hits the SIZE restriction which is 5, then the least used element is popped from the queue. This class has two public methods which are get and put.
      * A) get(string Key): Whenever a key is retrieved from the hash map, the element key inside the priority queue should be updated the current time stamp to indicate when it was last used.
      * B) put(string Key, String Value): This public method checks if the key already exists in the map and if it does then its time stamp gets updated with the new time when it was accessed. If it does not exist then just add the key to the priority queue.
      * C) There are 3 private methods which are insert, remove, and update which use PQ to perform tasks.
      * D) Update: The data structure for a priority queue can be a heap and it was assumed that modifying element and updating their timestamps in the priority queue would initiate re ordering each element after insertion. However, this was wrong since a heap can only update when inserting or removing an element and therefore, the element has to be deleted and then inserted through the partially sorted tree into the priority queue. Due to this, the update task is done in O(nlogn) complexity.

## **The Server:** Receiving requests from the client

* *Implementation:*
  + This is already explained here: [Server](#_Server:)

# **How to install and use my system**

1. Extract the contents from LopezAlejandroOS3.zip
2. Zip Folder contents:
   1. Operating Systems Project 3.docx: Design Document
   2. Output: Text files to be used
      1. Program1.txt
      2. Test.txt
      3. Test2.txt
   3. WordCount: Class files
      1. Application.java
      2. Cache.java
      3. Client.java
      4. Element.java
      5. Protocol. Java
      6. Server.java
3. Create a new Java project on the desired IDE (NetBeans, IntelliJ Idea, etc)
4. Inside the **src** folder create a new folder called WordCount (This is the package WordCount)
5. Inside the folder word count add the class files from Step 2 part C
6. Create another folder inside **src** called output and add the text files from Step 2 part B
7. Java Project should look like this:
   1. Graphical user interface

      Description automatically generated with medium confidence
8. Go to **Protocol.java** and change the path for the output folder to the one on your local machine:
   1. 
9. Open a PORT on your gateway or use one that is already open or currently not being used.
10. Go to **Server.java** and change the PORT to the one you need:
    1. 
11. Go to **Client.java** and once again change the PORT to yours
    1. Note: local host is used by default: **"127.0.0.1"**
12. Run **Server.java**
    1. 
13. Open as many instances of a command prompt with the same number of clients that you want to test
    1. 1 client -> 1 command prompt open
    2. 2 clients -> 2 command prompts open
14. Perform the following operations on the command prompt:
    1. Cd “Your WordCount Folder Path”
    2. Javac Client.java -> to compile the java class
    3. Java Client.java -> to run the java file
    4. The command prompt window(s) should look like this

Text

Description automatically generated

1. User input:
   1. List, Exit
      1. These commands only require one word with no extra spaces or anything else

Text

Description automatically generated

* 1. Create, Remove, Read:
     1. These commands have the structure as follows: Read Test.txt
        1. The first word is the command and followed by a space you have the File name that you want to work on

A screenshot of a computer

Description automatically generated with medium confidence

* 1. Update, Store
     1. These commands have the structure as follows:
        1. Update: Update Test.txt Tiger
           1. The first word is the command, the second is the file name, and the third is the value that you want the file to have
        2. Store: Store Test.txt Test2.txt
           1. Store copies the contents of one file to another. The first word is the command, the second the source file and the third is the destination file.

1. Opening more than one client:
   1. As mentioned before you just need to open multiple command prompts and access the same Client.java file. All clients can do operations even if there is more than one running.
2. Server output:
   1. Since the server is being ran through the IDE you can view the output on the Server.java being ran there, **However, after Steps 3-11 are completed you can open the Server.java through a command prompt as well.** And unlike the Client.java where you can open multiple clients, only one Server.java should be opened.
3. Important with multiple clients:
   1. The data store does get updated live but the system is structured so that when you update a file’s contents on one client it is not shown immediately on the other client. The client instance that changed the file first must read the file’s contents AFTER it updated them so that it can be showed on the other client too. Like this:

Text

Description automatically generated

* 1. On the picture:
     1. Client 1 is the bottom one and Client 2 is the one above
  2. Example:
     1. Client1:
        1. Read Test.txt
           1. Contents: Hi
        2. Update Test.txt hello
           1. Test.txt now has the value Hello
        3. Read Test.txt hello
           1. Contents: Hello
     2. Client2:
        1. Before the Read after updating on Client1:
           1. Read Test.txt

Contents: Hi

* + - 1. After the Read on Client1:
         1. Read Test.txt

Contents: Hello